SHARDA UNIVERSITY
Beyond Boundaries

Syllabus

UNIT-I: Basics of Python Programming

Introduction: A Brief History of Python, Applications areas of python, The Programming Cycle for Python.

Elements of Python: keywords and identifiers, variables, data types and type conversion, Indexing and Slicing, operators in python, Operator precedence and associativity, expressions in python.

Conditional Statements: if statement, if-else statement, Nested-if statement and el-if statements.

Loops: Purpose and working of loops, while loop, for loop, else with loop statement, Nested Loops, break, continue and pass statement.

Unit Objective

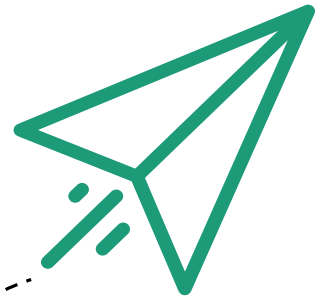
Understand the historical evolution and various applications of Python..

Familiarize with Python's programming cycle, keywords, variables, and operators.

Learn and apply conditional and loop statements in Python for controlling program flow.

Implement indexing, slicing, and manage operator precedence in Python code.

Gain proficiency in writing and debugging Python programs with decisions and loops.



Content(CO1)

INTRODUCTION

1.A Brief History of Python
2.Application Areas of Python
3.The Programming Cycle for Python

ELEMENTS OF PYTHON

1.Keywords and Identifiers
2.Variables
3.Data Types and Type Conversion
4.Indexing and Slicing
5.Operators in Python
   1. Types of Operators
   2. Operator Precedence and Associativity
6.Expressions in Python

CONDITIONAL STATEMENTS

1.if Statement
2.if-else Statement
3.Nested if Statement
4.elif Statement

LOOPS

1.Purpose and Working of Loops
2.while Loop
3.for Loop
4.else with Loop Statement
5.Nested Loops
6.Control Flow Statements: break, continue, and pass

A Brief History of Python

Creation: Python was created by **Guido van Rossum** in the late 1980s.

First Release: The first version of Python was released in **1991**.

Inspiration: Python was influenced by the **ABC programming language**, which emphasized readability and ease of use.

Name Origin: The language was named after the British comedy series **"Monty Python's Flying Circus"**, not the snake.

Key Features in Early Versions:

Exception handling
Functions and modules

A Brief History of Python

Python 2.0 (2000): Introduced significant features like:

- •List comprehensions
- •Garbage collection

Python 3.0 (2008): Major redesign with improvements, but introduced **incompatibilities** with Python 2, leading to a long transition period.

Current Popularity: Python is widely used due to its:

- •**Versatility** across different fields
- •Extensive **library support**
- •Application in **web development**, **data analysis**, **machine learning**, and **automation**.

Usage: Major companies like **Google**, **Facebook**, **Instagram**, and **Netflix** utilize Python in various applications.

Community and Growth: Python's active community continues to contribute to its development and widespread adoption.

Application Areas of Python

Web Development

1. Popular frameworks: **Django**, **Flask**
2. Used for backend development
3. Known for **rapid development** and **scalability**

Data Science and Analytics

1. Libraries: **Pandas**, **NumPy**, **Matplotlib**
2. Powerful for:
   - Data analysis
   - Data visualization
   - Statistical computing

Automation/Scripting

1. Used in **task automation** and **scripting**
2. Ideal for:
   - Web scraping (e.g., using BeautifulSoup)
   - Task scheduling
   - System administration

Machine Learning and AI

1. Libraries: **TensorFlow**, **Keras**, **Scikit-learn**
2. Applications:
   - Predictive modeling
   - Deep learning
   - Natural language processing (NLP)

Game Development

1. Libraries: **Pygame**
2. Useful for developing **simple games**
3. Ideal for prototyping and learning game development

GUI Application Development

1. Libraries: **Tkinter**, **PyQt**
2. Used for building **desktop applications** with graphical user interfaces

Application Areas of Python

Scientific Computing

1. Libraries: SciPy, SymPy
2. Widely used in fields like:
   - Physics
   - Biology
   - Mathematics

Cybersecurity

1. Python is used for:
   - Penetration testing
   - Network scanning (e.g., using Scapy)
   - Scripting exploits

Network Programming

1. Libraries: Socket programming, Twisted
2. Useful for building network applications like servers and clients

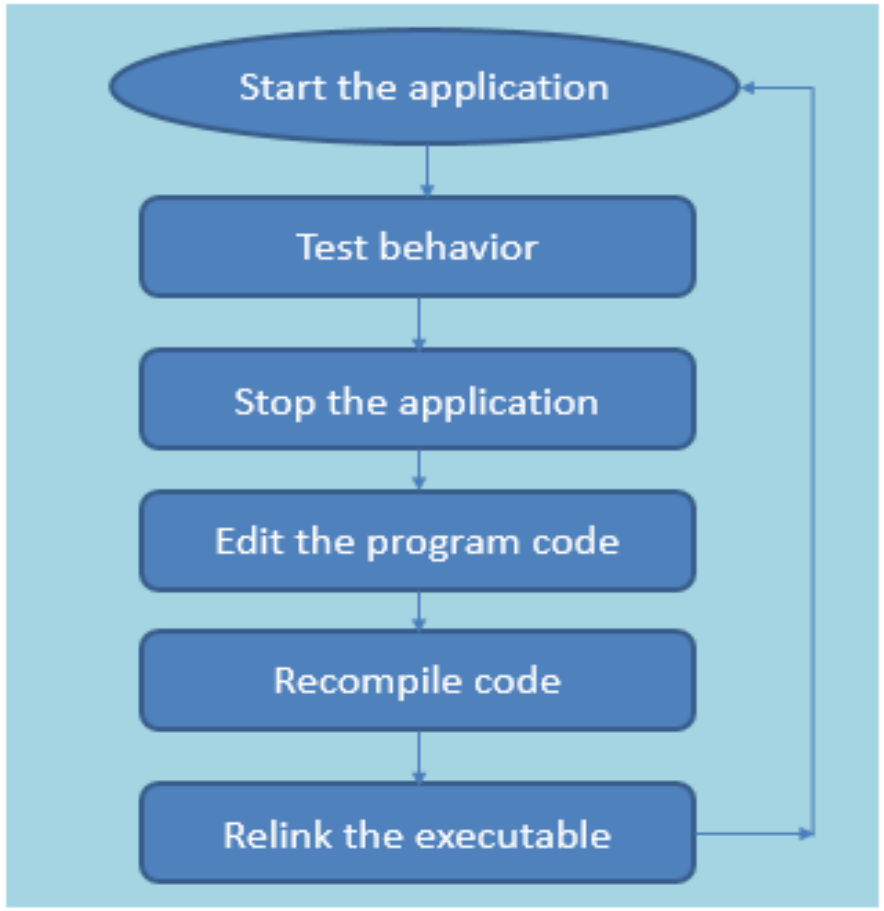
Internet of Things (IoT)

1. Python supports IoT devices through libraries like MicroPython, CircuitPython
2. Used in controlling sensors, devices, and building IoT systems

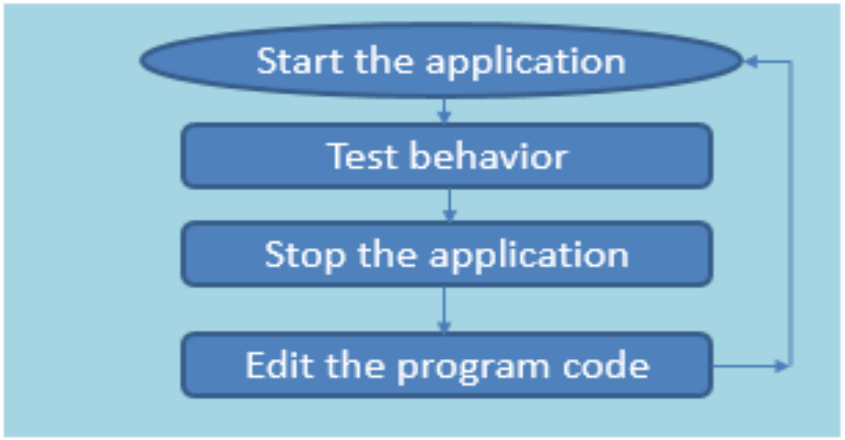
INTRODUCTION (CO1)

The Programming Cycle for Python

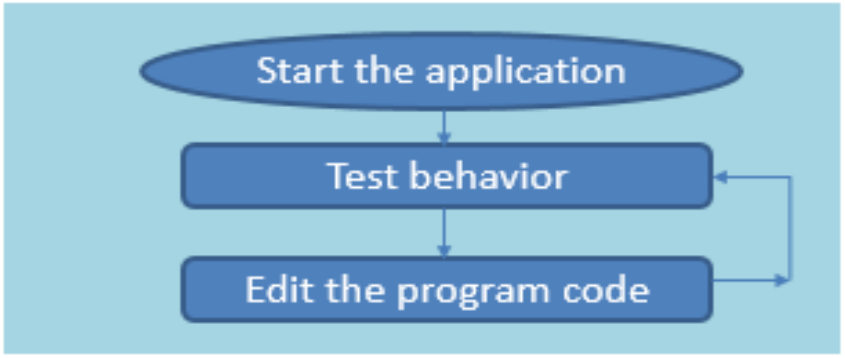
Traditional Development Cycle



Python's Development Cycle

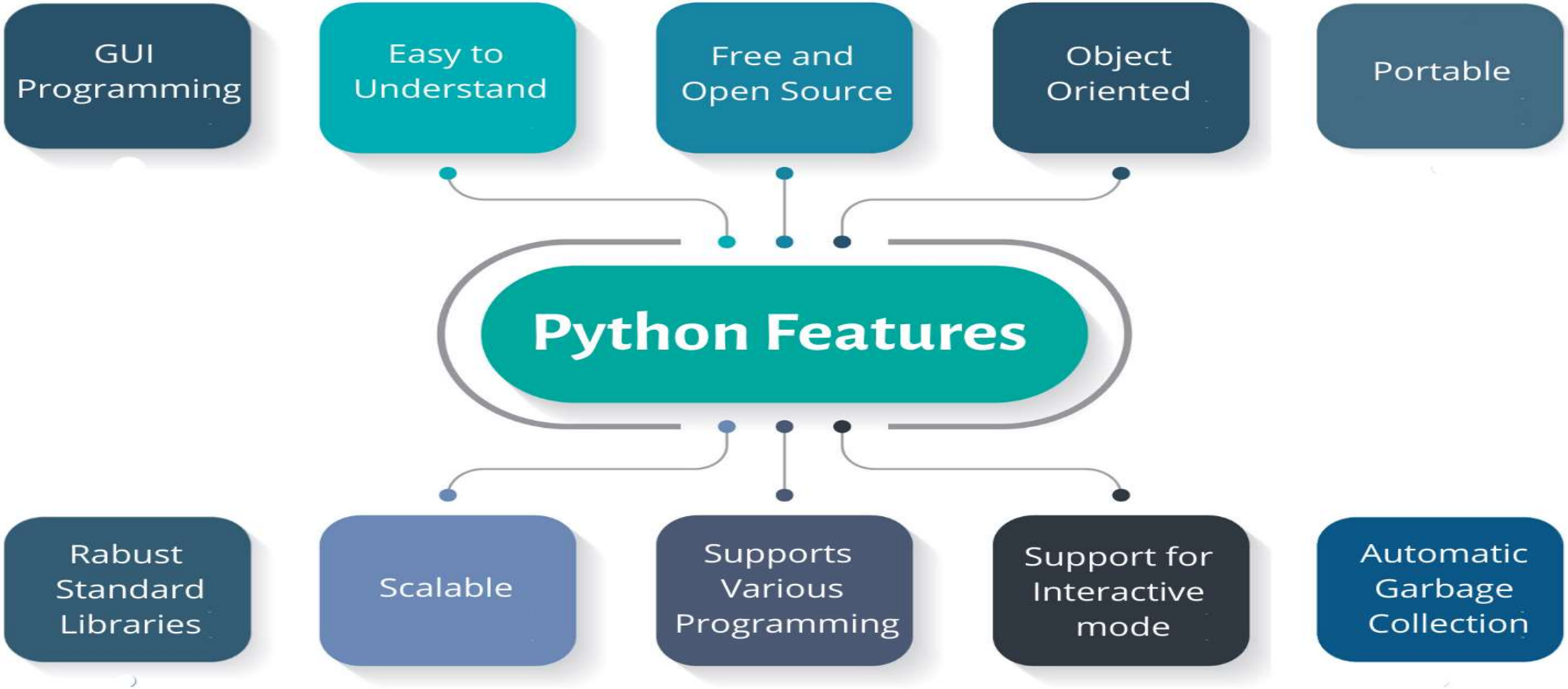


Python's Development Cycle with Module Reloading



INTRODUCTION (CO1)

Features of Python



ELEMENTS OF PYTHON (CO1)

Keywords

- **Definition**: Predefined, reserved words in Python.
- Cannot be used as **variable names**, **function names**, or **identifiers**.
- Python has **35 keywords** (as of Python 3.x).
- Common examples:
  - **Control flow keywords**: if, else, elif, for, while
  - **Function definition**: def, return
  - **Logical keywords**: and, or, not
  - **Others**: import, pass, yield , True, False, None, break

ELEMENTS OF PYTHON (CO1)

Identifiers

An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another.

Rules for writing Identifiers

- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore _.
- An identifier cannot start with a digit.
- Keywords cannot be used as identifiers.
- Cannot use special symbols like !, @, #, $, % etc. in identifier.
- An identifier can be of any length.

ELEMENTS OF PYTHON (CO1)

Identifiers Examples

Examples of Valid Identifiers

- studentName
- _employee_id
- num1
- calculate_sum

Examples of Invalid Identifiers

- 2ndValue (starts with a number)
- my-name (contains a hyphen)
- class (keyword used as an identifier)

ELEMENTS OF PYTHON (CO1)

Variables

- A variable is a location in memory used to store some data (value).
- Unique names are given to them to differentiate between different memory locations.
- No need to declare a variable before using it. The declaration happens automatically when a value is assigned to a variable.
- The Assignment operator (=) is used to assign values to variables.

ELEMENTS OF PYTHON (CO1)

Variables

- The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.
- For example

```
>>> num = 347              # An integer assignment
>>> x   = 45.89            # A floating point
>>> name   = “Aman"        # A string
```

ELEMENTS OF PYTHON (CO1)

Multiple Assignment in Variables

- A single value may be assigned to several variables simultaneously.
- For example

```
>>> a = b = c = 1
```

- Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location.
- It also allows to assign multiple objects to multiple variables.
- For example −

```
>>> a, b, c = 23, 29.45, “Ram”
```

- Here, two integer objects with values 23 and 29.45 are assigned to variables a and b respectively, and one string object with the value “Ram" is assigned to the variable c.

ELEMENTS OF PYTHON (CO1)

Data types

- Every value in Python has a datatype, that are used to define the operations possible on them and the storage method for each of them.

- Since everything is an object in Python programming, data types are classes and variables are instance (object) of these classes

ELEMENTS OF PYTHON (CO1)

Standard Data types

Data Types	Keyword
Text Type	str
Numeric Types	int, float, complex
Sequence Types	list, tuple, range
Mapping Type	dict
Set Types	set
Boolean Types	bool
Binary Types	bytes

ELEMENTS OF PYTHON (CO1)

Numeric Data types

Python supports several numeric types:

1.int: Integer numbers (e.g., 5, -10)
2.float: Floating-point numbers (e.g., 3.14, -0.001)
3.complex: Complex numbers (e.g., 2 + 3j)

Integer (int)
Definition: Whole numbers, both positive and negative, without a decimal point.
Example:

- a = 10
- b = -20

No limit on the size of integers in Python.

Floating-point Numbers (float)
Definition: Numbers with decimal points or in exponential notation.
Example:

- pi = 3.14159
- exp = 2e5 (which is 2 × 10^5 or 200,000)

Used for **real numbers** (fractions).

Complex Numbers (complex)
Definition: Numbers in the form of **a + bj**, where a is the real part and b is the imaginary part.
Example:

z = 2 + 3j
real_part = z.real → returns 2
imaginary_part = z.imag → returns 3

Supports complex arithmetic.

ELEMENTS OF PYTHON (CO1)

Strings Data types

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

Strings Data types

- For example

```
>>> str = 'Hello World!'
>>> print(str)          # Prints complete string
>>> print(str[0])       # Prints first character of the string
>>> print(str[2:5])     # Prints characters starting from 3rd to 5th
>>> print(str[2:])      # Prints string starting from 3rd character
>>> print(str * 2)      # Prints string two times
>>> print(str + 'TEST') # Prints concatenated string
```

List Data types

- A List is an ordered sequence of item, It contains items separated by commas and enclosed within square brackets ([]).
- To some extent, lists are like arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
- Declaring a list is straight forward. Items separated by commas are enclosed within brackets [ ].

ELEMENTS OF PYTHON(CO1)

List Data types

- >>> a = [1, 2.2, 'python']
  - The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

ELEMENTS OF PYTHON(CO1)

List Data types

```
>>>  list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
>>> tinylist = [123, 'john']
    >>> print(list)                    # Prints complete list
    >>> print(list[0])                 # Prints first element of the list
    >>> print(list[1:3])               # Prints elements starting from 2nd till 3rd
    >>> print(list[2:])                # Prints elements starting from 3rd element
    >>> print(list * 2)                # Prints list two times
    >>> print(list + tinylist)         # Prints concatenated lists
```

ELEMENTS OF PYTHON(CO1)

Tuple Data types

- A tuple is similar to the list. A tuple consists of several values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated.
- Tuples can be thought of as read-only lists.
- The only difference is that tuples are immutable. Tuples once created cannot be modified.
- Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically.

ELEMENTS OF PYTHON(CO1)

Tuple Data types

```
>>> tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
>>> tinytuple = (123, 'john')
    >>> print(tuple)                    # Prints complete tuple
    >>> print(tuple[0])                 # Prints first element of the tuple
    >>> print(tuple[1:3])               # Prints elements starting from 2nd till 3rd
    >>> print(tuple[2:])                # Prints elements starting from 3rd element
    >>> print(tuple * 2)                # Prints tuple two times
    >>> print(tuple + tinytuple)        # Prints concatenated tuples
```

ELEMENTS OF PYTHON(CO1)

Set Data types

- Set is an unordered collection of unique items.
- Set is defined by values separated by comma inside braces { }.
- Items in a set are not ordered.
- Set operations can be performed like union, intersection on two sets. Set have unique values. They eliminate duplicates.
- indexing has no meaning as set are unordered collection. Hence the slicing operator [] does not work.
- For example

```
>>>a = {1,2,2,3,3,3}
>>> a
{1, 2, 3}
```

Dictionary Data types

- Dictionary is an unordered collection of key-value pairs.
- It is generally used when we have a huge amount of data.
- Dictionaries are optimized for retrieving data. It is must to know the key to retrieve the value.
- A dictionary key can be almost any Python type but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
- Dictionaries are defined within braces {} with each item being a pair in the form key: value. Key and value can be of any type.
- For example

```
>>> d = {1:'value','key':2}
>>> type(d)
<class 'dict'>
```

- **Variable Scope**: Variables have a scope, which defines where they can be accessed from. Variables declared inside a function have local scope, meaning they are only accessible within that function. Variables declared outside of any function have global scope and can be accessed from anywhere in the code.

```
x = 10  # global variable


def my_function():
    y = 20  # local variable
    print(x)  # x can be accessed here
    print(y)  # y can be accessed here


my_function()
print(x)  # x can be accessed here
print(y)  # Error: y is not defined
```

Variables are fundamental to programming in Python, as they allow you to store and manipulate data within your programs.

Python Console input()

- In Python, you can use the input() function to read input from the console (also known as standard input). Here's how you can use it:

 # Prompt the user for input and store it in a variable

```
user_input = input("Enter your name: ")
```

 # Print the input provided by the user

```
print("Hello,", user_input)
```

When you run this code, it will display "Enter your name: " in the console, and wait for the user to input something. After the user presses Enter, the input will be stored in the variable user_input, and then the code will print "Hello, " followed by whatever the user entered.

You can use the input() function without any arguments as well, which will simply wait for the user to enter something without displaying any prompt:

```
user_input = input()
print("You entered:", user_input)
```

Keep in mind that the input() function always returns a string, even if the user inputs a number or other data type. If you need to convert the input to a different data type, you can use type casting:

```
user_input = input("Enter a number: ")

# Convert the input to an integer
number = int(user_input)

# Print the input as an integer
print("You entered:", number)
```

Numerical Input

To read numerical input from the console in Python, you can use the input() function to capture the input as a string, and then convert it to the desired numerical data type using type casting. Here's how you can do it:

```
# Read numerical input from the console
user_input = input("Enter a number: ")
 # Convert the input to an integer using int()
number = int(user_input);    print("You entered an integer:", number)
```

In this example, the input() function prompts the user to enter a number, and whatever the user enters is captured as a string in the user_input variable. Then, we use the int() function to convert this string to an integer. If the user enters a value that cannot be converted to an integer (e.g., a string or a floating-point number), a ValueError will be raised. We handle this exception using a try-except block to provide feedback to the user.

Similarly, if you want to read floating-point numbers from the console, you can use the float() function for conversion:

```
# Read numerical input from the console

user_input = input("Enter a floating-point number: ")

 # Convert the input to a float using float()

    float_number = float(user_input)

    print("You entered a float:", float_number)
```

ELEMENTS OF PYTHON(CO1)

Type conversion

- The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion.
- Python has two types of type conversion.
  - Implicit Type Conversion
  - Explicit Type Conversion

ELEMENTS OF PYTHON(CO1)

Implicit Type conversion

- Python automatically converts one data type to another data type without any user involvement.
- It always converts smaller data types to larger data types to avoid the loss of data.
- Example

```
>>> a = 4
>>> b = 2.3
>>> c = a + b
```

- In above example, data type of a and b are int and float, respectively. Data type of c will be float and its value is 6.3

ELEMENTS OF PYTHON(CO1)

Explicit Type conversion

- In Explicit Type Conversion, users convert the data type of an object to required data type.
- The predefined functions like int(), float(), str() are used to perform explicit type conversion.
- This type of conversion is also called typecasting because the user casts/changes the data type of the objects.
- Syntax

  <datatype>(expression)
- Example

  >>> a = 2.6

  >>> c = int(a)

ELEMENTS OF PYTHON(CO1)

Function of type conversion

Function	Description
int(x)	Convert x to an integer
float(x)	Convert x to a float number
str(x)	Convert x to a string
tuple(x)	Convert x to a tuple
list(x)	Convert x to a list
set(x)	Convert x to a set
ord(x)	Convert x to ASCII code
bin(x)	Convert x to a binary
oct(x)	Convert x to an octal
hex(x)	Convert x to a hexadecimal
chr(x)	Convert x to a character

Working with Complex numbers

Complex Number Attributes

Attribute	Description
num.real	Real component of complex number num
num.imag	Imaginary component of complex number num
num.conjugate()	Returns complex conjugate of num

```
aComplex = -8.3 - 1.4j
print aComplex
print aComplex.real
print aComplex.imag
print aComplex.conjugate()
```

Working with Complex numbers

Use abs(z) to get its magnitude (as a float) or z.real to get its real part : complex

```
a=1.5+0.5j

print a.real
print abs(a)
```

ELEMENTS OF PYTHON(CO1)

Operators in Python

Python language supports the following types of operators.

- Arithmetic Operators
- Relational Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

ELEMENTS OF PYTHON(CO1)

Arithmetic Operators

Operator	Description	Example (a=5, b=3)
+	Adds values on either side of the operator.	a + b = 8
-	Subtracts right hand operand from left hand operand.	a – b = 2
*	Multiplies values on either side of the operator	a * b = 15
/	Divides left hand operand by right hand operand	a / b = 1.6667
%	Divides left hand operand by right hand operand and	a % b = 2
**	Performs exponential (power) calculation on operators	a**b =5^3= 125
//	The division of operands where the result is the quotient in which the digits after the decimal point are removed.	a//b=1

ELEMENTS OF PYTHON(CO1)

Relational Operators

Operator	Description	Example a=5, b=3
==	If the values of two operands are equal, then the condition becomes true	a == b is not true.
!=	If values of two operands are not equal, then condition becomes true.	a!=b is true
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	a > b is true
<	If the value of left operand is less than the value of right operand, then condition becomes true.	a < b is not true
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	a >= b is true
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	a <= b is not true

ELEMENTS OF PYTHON(CO1)

Assignment Operators

Operator	Description
=	a = b Assigns values from right side operands(b) to left side operand (a)
+=	a+=b is same as a = a + b It adds right operand to the left operand and assign the result to left operand
-=	a-=b is same as a = a - b It subtracts right operand from the left operand and assign the result to left operand
*=	a*=b is same as a = a * b It multiplies right operand with the left operand and assign the result to left operand
/=	a/=b is same as a = a / b It divides left operand with the right operand and assign the result to left operand

ELEMENTS OF PYTHON(CO1)

Assignment Operators

Operator	Description
%=	a%=b is same as a = a % b It takes modulus using two operands and assign the result to left operand
**=	a**=b is same as a = a ** b Performs exponential (power) calculation on operators and assign value to the left operand
//=	a//=b is same as a = a //b It performs floor division on operators and assign value to the left operand

ELEMENTS OF PYTHON(CO1)

Logical Operators

Operator	Description
and	True if both the operands are true
or	True if either of the operands is true
not	True if operand is false (complements the operand)

ELEMENTS OF PYTHON(CO1)

Logical Operators

a	b	a and b	a or b	not a
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Bitwise Operators

- Bitwise operators manipulate the data at bit level.
- These are applicable on integer values.
- Types
  - & (Bitwise and operator)
  - | (Bitwise or operator)
  - ^ (Bitwise XOR operator)
  - ~ (Bitwise one's complement operator)
  - << (Bitwise left-shift operator)
  - >> (Bitwise right-shift operator)

ELEMENTS OF PYTHON(CO1)

Bitwise Operators

a	b	a & b	a | b	a ^ b	~a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

ELEMENTS OF PYTHON(CO1)

Bitwise Operators

Operator	**Let a = $(92)_{10}$ =$(0101\ 1100)_2$ and b = $(14)_{10}$ = $(0000\ 1110)_2$**	
&	c = (a & b) = 92 & 14	0101 1100 & 0000 1110 ---------- 0000 1100 = $(12)_{10}$
|	c = (a | b) = 92 | 14	0101 1100 | 0000 1110 ---------- 0101 1110 = $(94)_{10}$
^	c = (a ^ b) = 92 | 14	0101 1100 ^ 0000 1110 ---------- 0101 0010 = $(82)_{10}$

ELEMENTS OF PYTHON(CO1)

Bitwise Operators

Operator	**Let a = $(92)_{10}$ =$(0101\ 1100)_2$ and b = $(14)_{10}$ = $(0000\ 1110)_2$**	
~	c = ~a = ~92	c = ~(0101 1100) = 1010 0011
<<	c = a<< 1 = 46<< 1	C = 00101 110 << 1 = 01011100 = $(92)_{10}$
>>	c = a >> 2 = 92>> 2	C = 0101 1100 >> 2 = 0001 0111 = $(23)_{10}$

ELEMENTS OF PYTHON(CO1)

Membership Operators

- **in** and **not in** are the membership operators in Python.
- They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).
- In a dictionary, we can only test for presence of key, not the value.

Operator	Description	Example x = {2,3,5}	Result
in	True if value/variable is found in the sequence	5 in x	True
not in	True if value/variable is not found in the sequence	5 not in x	False

ELEMENTS OF PYTHON(CO1)

Identity Operators

- Identity operators compare the memory locations of two objects.
- **is** and **is not** are the identity operators in Python.
- They are used to check if two values (or variables) are located on the same part of the memory.
- Two variables that are equal does not imply that they are identical.

Operator	Description	Example a = 'Hello' b = 'Hello'	Result
is	True if the operands are identical (refer to the same object)	a is b	True
is not	True if the operands are not identical (do not refer to the same object)	a is not b	False

ELEMENTS OF PYTHON(CO1)

Operator Precedence and associativity

- When an expression contains two or more than two operators, then it is evaluated based on operator precedence and associativity.
- Each operator of same precedence is grouped in same level and operator of higher precedence is evaluated first.
- Two or more operators having equal precedence are evaluated either left-to-right(LTR) or right-to-left(RTL) based on their associativity.

ELEMENTS OF PYTHON(CO1)

Operator Precedence and associativity

- When an expression contains two or more than two operators, then it is evaluated based on operator precedence and associativity.
- Each operator of same precedence is grouped in same level and operator of higher precedence is evaluated first.
- Two or more operators having equal precedence are evaluated either left-to-right(LTR) or right-to-left(RTL) based on their associativity.

ELEMENTS OF PYTHON(CO1)

Operator Precedence and associativity

Operator	Description
()	Parenthesis
**	Exponentiation
~,+,-	Unary operators
*,/,//,%	Arithmetic multiply, division, floor and modulo division
+, -	Addition and subtraction
>>,<<	Bitwise left and right shift operator
&	Bitwise and operator
^	Bitwise Ex-or operator
|	Bitwise or operator

ELEMENTS OF PYTHON(CO1)

Operator Precedence and associativity

Operator	Description
<=,>=,<,>	Relational inequality operators
==, !=	Equal and not equal operators
=,*=,/=,//=,%=,+=,-=,**=,&=	Assignment operators
is, is not	Identity operators
in, not in	Membership operators
not	Logical not operator
and	Logical and operator
or	Logical or operator

ELEMENTS OF PYTHON(CO1)

Expressions in Python

- Expressions are representations of value.
- They are different from statement in the fact that statements do something while expressions are representation of value.
- Python expression contains identifiers, literals, and operators.
- For Example

  x = a*b + c/d –f is an expression.

ELEMENTS OF PYTHON(CO1)

Expressions in Python

- Expressions are representations of value.
- They are different from statement in the fact that statements do something while expressions are representation of value.
- Python expression contains identifiers, literals, and operators.
- For Example

  x = a*b + c/d –f is an expression.

ELEMENTS OF PYTHON(CO1)

MCQ

1. What is the correct file extension for Python files?
   a) .pyt
   b) .py
   c) .pt
   d) .pyth
2. What is the output for –

   'python ' [-3]?
   a) 'o'
   b) 't'
   c) 'h'
   d) Negative index error.

ELEMENTS OF PYTHON(CO1)

MCQ

3. How do you create a variable with the numeric value 5?
   a) x = 5
   b) x = int(5)
   c) Both option (a) and (b) are correct
   d) No option is correct
4. Which one is NOT a legal variable name?
   a) My_var
   b) My-var
   c) _Myvar
   d) Myvar

ELEMENTS OF PYTHON(CO1)

MCQ

3. How do you create a variable with the numeric value 5?
   a) x = 5
   b) x = int(5)
   c) Both option (a) and (b) are correct
   d) No option is correct
4. Which one is NOT a legal variable name?
   a) My_var
   b) My-var
   c) _Myvar
   d) Myvar

ELEMENTS OF PYTHON(CO1)

MCQ

5. Which operator can be used to compare two values?
   a) <>
   b) ><
   c) ==
   d) =

6. Which one is a legal identifier?
   a) ab#cd
   b) abc
   c) S.I
   d) Area of circle

ELEMENTS OF PYTHON(CO1)

MCQ

7. Which of these collections defines a SET?
   a) {1, 2, 3, 4}
   b) {‘1’:’12’,’2’:’45’}
   c) (1,2,3,4)
   d) [1,2,3,4]

ELEMENTS OF PYTHON(CO1)

MCQ

7. Which of these collections defines a SET?
   a) {1, 2, 3, 4}
   b) {‘1’:’12’,’2’:’45’}
   c) (1,2,3,4)
   d) [1,2,3,4]

CONDITIONAL STATEMENTS(CO1)

Definition: Conditional statements control the flow of a program by executing certain blocks of code only if specific conditions are met.

In Python, conditional statements include:

- if
- if-else
- elif
- Nested if

•.

SHARDA UNIVERSITY
Beyond Boundaries

Topic Objective (CO1)

Develop expertise in Python's control structures, loops and troubleshooting decision-driven programs

CONDITIONAL STATEMENTS(CO1)

if Statement

- if Statement is used to run a statement conditionally i.e. if given condition is true then only the statement given in if block will be executed.
- An if statement consists of a Boolean expression followed by one or more statements.
- Syntax:

```
if Boolean Expression:
        Statements
```

- Example:

```
age = int(input("Enter your age in years: "))
if age >= 18:
   print("You can VOTE")
   print("Congrats!“)
```

CONDITIONAL STATEMENTS(CO1)

Important points about Statement

- The colon (:) is significant and required. It separates the **header** of the **compound statement** from the **body**.
- The line after the colon must be indented. It is standard in Python to use four spaces for indenting.
- All lines indented the same amount after the colon will be executed whenever the Boolean expression is true.

Example:

```
x = 5
if x > 0:
    print("x is positive")
```

CONDITIONAL STATEMENTS(CO1)

if else Statement

- An if statement can be followed by an optional else statement, which executes when the Boolean expression is *FALSE*.
- Syntax:

```
if Boolean Expression:
        Statement-A
else:
        Statement-B
```

- Example:

```
percentage = int(input("Enter your percentage: "))
if percentage > 33:
   print("PASS")
else:
   print("FAIL")
```

CONDITIONAL STATEMENTS(CO1)

if elif Statement

- The **if elif** statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.
- Unlike **else**, for which there can be at most one statement, there can be an arbitrary number of **elif** statements following an **if**.
- Syntax:

```
if Boolean Expression1:
        Statement-A
elif Boolean Expression2:
        Statement-B
elif Boolean Expression3:
        Statement-C
else:
        Statement-D
```

CONDITIONAL STATEMENTS(CO1)

if elif Statement Example

```
percentage = int(input("Enter your percentage: "))
if percentage >= 90:
    print("Grade A")
elif percentage >= 80:
    print("Grade B")
elif percentage >= 70:
    print("Grade C")
elif percentage >= 60:
    print("Grade D")
elif percentage >= 50:
    print("Grade E")
else:
    print("Fail")
```

CONDITIONAL STATEMENTS(CO1)

Nested if else Statement

- One conditional can also be nested within another.
- You can use one if or else if statement inside another if or else.
- Syntax:

```
if Boolean Expression1:
        Statement-A
else:
        if Boolean Expression-2:
                Statement-B
        else:
                Statement-C
```

CONDITIONAL STATEMENTS(CO1)

Nested if else Statement Example

```
a = int(input("Enter first Number"))
b = int(input("Enter Second Number"))
c = int(input("Enter third Number"))
if a > b:
    if a > c:
        print(a, "is greatest")
    else:
        print(b, "is greatest")
else:
    if b > c:
        print(b, "is greatest")
    else:
        print(c, "is greatest")
```

LOOPS(CO1)

Definition: Loops allow us to execute a block of code **repeatedly** based on a condition.

Two main types of loops in Python:

1.while loop
2.for loop
3.Nested loop statement

Why use loop statements?

- Program statements are executed sequentially one after another.
- In some situations, a block of code needs to be executed multiple times.
- To achieve this, we use looping statements in Python programming

while Loop

A while loop statement in Python programming language repeatedly execute a target statement as long as a given condition is true.

Syntax:

```
while Expression:
        Statements
```

LOOPS(CO1)

Program to find the sum of first n natural numbers

```
num = int(input("Enter a number: "))
sum_n = 0
while num > 0:
    sum_n = sum_n + num
    num = num - 1
print("The Sum is: ", sum_n)
```

Output:

Enter a number: 4

The sum is: 10

LOOPS(CO1)

for in Loop

- For loops are used for sequential traversal. For example: traversing a list or string or array etc.
- It can be used to iterate over a range and iterators(list,set,tuple etc).

- **Syntax:**

```
for value in sequence:
        Statements
```

Important points about “for in” loop syntax

- Here, ***value*** is the variable that takes the value of the item inside the sequence on each iteration.
- Loop continues until we reach the last item in the sequence.
- The body of for in loop is separated from the rest of the code using indentation.

LOOPS(CO1)

Program to find the sum of all numbers stored in a list

```
numbers = [10, 20, 5, 15, 25, 35, 30]
sum_num = 0
for val in numbers:
    sum_num = sum_num + val
print("The sum of numbers is:", sum_num)
```

Output:

The sum of numbers is: 140

LOOPS(CO1)

Program to find the sum of all numbers stored in a list

```
numbers = [10, 20, 5, 15, 25, 35, 30]
sum_num = 0
for val in numbers:
    sum_num = sum_num + val
print("The sum of numbers is:", sum_num)
```

Output:

The sum of numbers is: 140

range() function

- We can generate a sequence of numbers using range() function.
- range(10) will generate numbers from 0 to 9 (10 numbers).
- range() function returns the list, all the operations that can be applied on the list can be used on it.
- We can also define the start, stop and step size as
  - ***range(start, stop, step_size).***
  - step_size defaults to 1 if not provided.

LOOPS(CO1)

Example of range() function

```
for i in range(0, 100, 25):
    print(i)
```

Output:

0

25

50

75

Nested loop Example

Python programming language allows to use one loop inside another loop.

Syntax:

```
for value1 in sequence:
        for value2 in sequence:
                statement(s)
        statement(s)
```

Nested loop Example

```
for i in range(1, 5):
    for j in range(i):
        print(i, end=" ")
    print()
```

Output:

1

2 2

3 3 3

4 4 4 4

LOOPS(CO1)

Program to print all prime numbers from a list

```
n = [2, 4, 5, 6, 8, 11, 14, 25, 27, 31]
for x in n:
    i = 1
    flag = 0
    while i <= x:
        if x % i == 0:
            flag = flag + 1
        i = i + 1
    if flag == 2:
        print(x)
```

Output: 2 5 11 31

break statement

- The break statement is used to exit from a for in or a while loop.
- The purpose of this statement is to end the execution of the loop (for or while) immediately and the program control goes to the statement after the last statement of the loop.
- If there is an optional else statement in while or for loop it skips the optional clause also.
- Syntax:

```
for variable in sequence:
        statement_1
        statement_2
        if expression3 :
                break
```

LOOPS(CO1)

Example of break Statement

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
num_sum = 0
count = 0
for x in numbers:
    num_sum = num_sum + x
    count = count + 1
    if count == 5:
        break
print("Sum of first ", count, "integers is: ", num_sum)
```

Output:

Sum of first 5 integers is: 15

continue statement

- The continue statement is used in a while or for in loop to take the control to the top of the loop without executing the rest statements inside the loop.
- Syntax:

```
for variable in sequence:
        statement_1
        statement_2
        if expression3 :
                continue
```

Example of continue Statement

```
for x in range(7):
    if x == 3 or x == 6:
        continue
    print(x)
```

Output:

0

1

2

4

5

pass statement

- In Python programming, the pass statement is a null statement.
- The difference between a comment and a pass statement in Python is that while the interpreter ignores a comment entirely, pass is not ignored.
- However, nothing happens when the pass is executed. It results in no operation.
- Syntax:

```
if Boolean expression :
        pass
else:
        Statements
```

Example of continue Statement

```
for x in range(7):
    if x == 3 or x == 6:
        continue
    print(x)
```

Output:

0

1

2

4

5

Example of pass Statement

```
number = int(input("Enter a number: "))
if number is 2:
    pass
else:
    print("Number is: ", number)
```

Output:

Number is: 5

Quiz (CO1)

- What is the output of the following?

```
i = 2
while True:
    if i % 3 == 0:
        break
    print(i)
    i += 2
```

Output: 2

4

- What is the output of the following?

```
x = "abcd"
    for i in x:
        print(i.upper(), end=" ")
```

Output: A B C D

- How many times the following loop will run?

```
for i in range(-3):
    print(i)
```

Output: 0 times

Quiz (CO1)

- How many times following loop will run.

  for i in [1, 2, 3]:

  a) 0
  b) 1
  c) 3
  d) 2

- for loop in python works on –
  a) range
  b) iteration
  c) Both the above
  d) None of above

- To break the infinite loop, which keyword is used in python?
  a) break
  b) exit
  c) continue
  d) None of above

- What will be the final value of i after

  for i in range(3):

  a) 1
  b) 2
  c) 0
  d) 3

Weekly Assignment(CO1)

- Write a program to find sum and reverse of digits in a number entered by the user(both within same loop).
- Write a program to find sum of terms of a Fibonacci series upto n terms.
- Write a program to find sum of all prime numbers and composite numbers separately within a given range.
- Write Programs to print following patterns.

a.
```
*
**
***
****
```

b.
```
*
**
***
****
```

c.
```
   *
  ***
 *****
*******
```

d.
```
   *
  ***
 *****
  ***
   *
```

e.
```
*****
****
***
**
*
```

f.
```
    *****
     ****
      ***
       **
        *
```

g.
```
**********
*        *
*        *
*        *
**********
```

h.
```
  *********
 *       *
  * * *
  * * *
   * *
```

Old Question Papers

- Explain the purpose and working of loops. Discuss break and continue with example. Write a python program to convert time from 12-hour format to 24-hour format. [AKTU 2019-20 (Odd) Marks-10]
- Write Python code to find the factorial of a number. [NIET Autonomous 2020-21 (Odd) Marks-06]
- Write Python program to convert uppercase letter to lowercase and vice versa. [NIET Autonomous 2020-21 (Odd) Marks-06]
- Explain the purpose and working of loops. Discuss Break and continue. [NIET Autonomous 2019-20 (Odd) Marks-10]
- Write Python Programs to print following patterns. [NIET Autonomous 2020-21 (Odd) Marks-10]

```
1                    *
010                  ***
10101                *****
0101010              *******
```

Expected Questions for University Exam(CO2)

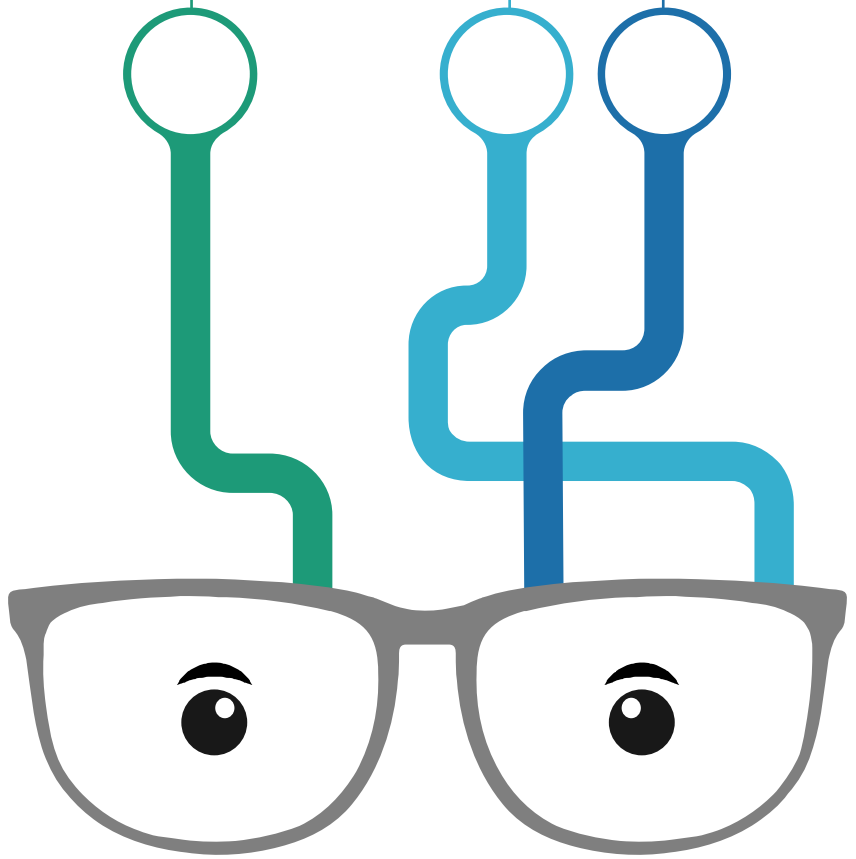
- Discuss functions in python with its part and scope. Explain with example(Take simple calculator with add , subtract , multiplication and division.)
- Explain higher order functions with respect to lambda expressions. Write a python code to count occurrences of an element in the list.
- Write a program to sort list of dictionaries by value in python-using lambda functions.
- Differentiate iterators and recursion. Write a program for recursive Fibonacci series.

Summary(CO1)

A function is a block of organized, reusable code that is used to perform a single, related action.

A module is a Python object with arbitrarily named attributes that can bind and reference.

Package can be installed for system wide use by running the setup script.

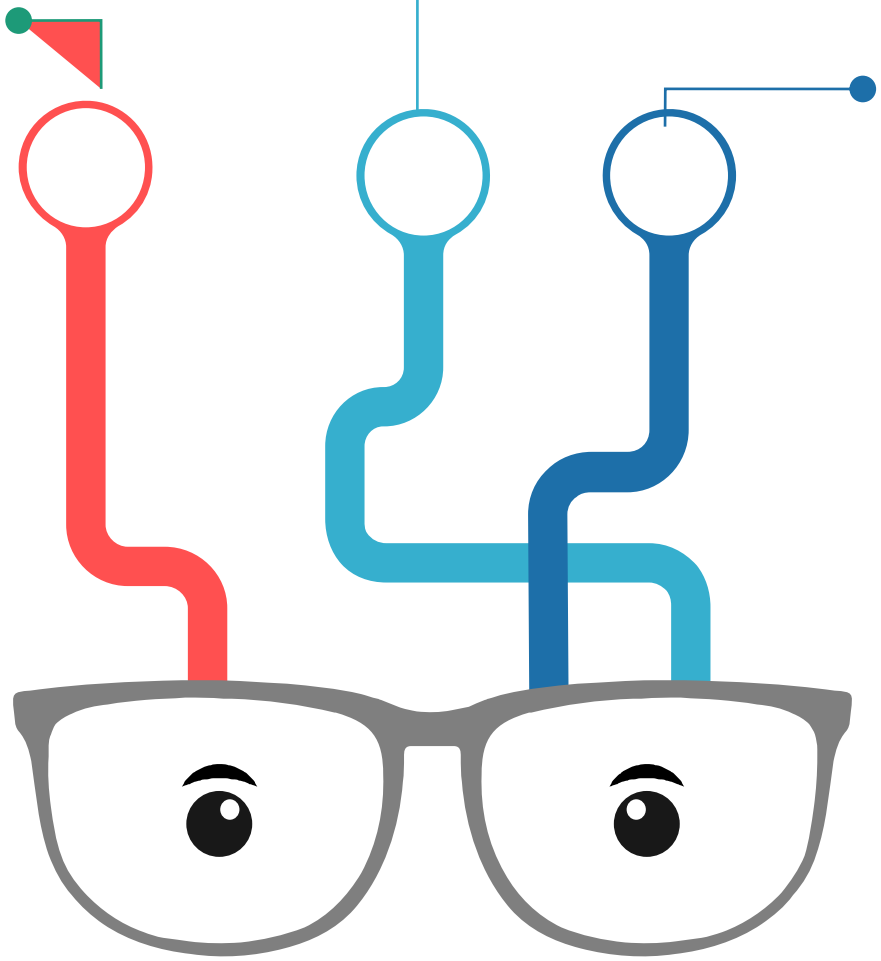


Summary(CO1)

Functional programming in Python provides powerful tools for writing concise, expressive, and efficient code.

Key concepts include closures, which capture external variables, decorators for modifying function behavior, and generators for memory-efficient iteration.

Co-routines allow functions to pause and resume, making them useful for handling asynchronous tasks..



References(CO2)

- "Python Programming: A Modern Approach" by Vamsi Kurama
- “Core Python Programming" by Dr. R. Nageswara Rao
- “Python Programming for Beginners" by Balagurusamy
- “Problem Solving and Python Programming" by Ashok Namdev Kamthane and Amit Ashok Kamthane
- “Introduction to Computing and Problem Solving with Python" by Jeeva Jose and P. Sojan Lal
- www.python.org
- www.w3schools.com

TEXT & REFERENCE BOOKS

Text books:

(1) Magnus Lie Hetland, "Beginning Python-From Novice to Professional"—Third Edition, Apress

(2) Peter Morgan, Data Analysis from Scratch with Python, AI Sciences

(3) Allen B. Downey, “Think Python: How to Think Like a Computer Scientist”, 2nd edition, Updated for Python 3, Shroff/O‘Reilly Publishers, 2016

(4) Miguel Grinberg, Developing Web applications with python, OREILLY

Reference Books:

(1) Dusty Phillips, Python 3 Object-oriented Programming - Second Edition, O’Reilly

(2) Burkhard Meier, Python GUI Programming Cookbook - Third , Packt

(3) DOUG HELLMANN, THE PYTHON 3 STANDARD LIBRARY BY EXAMPLE, :Pyth 3 Stan Libr Exam _2 (Developer's Library) 1st Edition, Kindle Edition.

(4) Kenneth A. Lambert, ―Fundamentals of Python: First Programs‖, CENGAGE Learning, 2012.